Rewriting modulo in Deduction modulo

Frédéric Blanqui

Rewriting modulo in Deduction modulo

Frédéric Blanqui

Laboratoire d'Informatique de l'École Polytechnique
91128 Palaiseau Cedex, France

Abstract. We study the termination of rewriting modulo a set of equations in the Calculus of Algebraic Constructions, an extension of the Calculus of Constructions with functions and predicates defined by higher-order rewrite rules. In a previous work, we defined general syntactic conditions based on the notion of computability closure for ensuring the termination of the combination of rewriting and β-reduction.

Here, we show that this result is preserved when considering rewriting modulo a set of equations if the equivalence classes generated by these equations are finite, the equations are linear and satisfy general syntactic conditions also based on the notion of computability closure. This includes equations like associativity and commutativity and provides an original treatment of termination modulo equations.

1 Introduction

The Calculus of Algebraic Constructions (CAC) [2, 3] is an extension of the Calculus of Constructions (CC) [9] with functions and predicates defined by (higher-order) rewrite rules. CC embodies in the same formalism Girard's polymorphic λ-calculus and De Bruijn's dependent types, which allows one to formalize propositions and proofs of (impredicative) higher-order logic. In addition, CAC allows functions and predicates to be defined by any set of (higher-order) rewrite rules. And, in contrast with (first-order) Natural Deduction Modulo [13], proofs are part of the terms.

Very general conditions are studied in [2, 4] for preserving the decidability of type-checking and the logical consistency of such a system. But these conditions do not take into account rewriting modulo equations like associativity and commutativity (AC), which would be very useful in proof assistants like Coq [22] since it increases automation and decreases the size of proofs. We already used the rewriting engine of CiME [8], which allows rewriting modulo AC, for a prototype implementation of CAC, and now work on a new version of Coq including rewriting modulo AC. In this paper, we extend the conditions given in [2] to deal with rewriting modulo equations.

2 The Calculus of Algebraic Constructions

We assume the reader familiar with typed λ-calculi [1] and rewriting [11]. The Calculus of Algebraic Constructions (CAC) [2] simply extends CC by considering a set $\mathcal{F}$ of *symbols* and a set $\mathcal{R}$ of *rewrite rules*. The terms of CAC are:

$$t, u \in \mathcal{T} ::= s \mid x \mid f \mid [x : t]u \mid tu \mid (x : t)u$$

where $s \in \mathcal{S} = \{\star, \Box\}$ is a *sort*, $x \in \mathcal{X}$ a *variable*, $f \in \mathcal{F}$, $[x : t]u$ an *abstraction*, tu an *application*, and $(x : t)u$ a *dependent product*, written $t \Rightarrow u$ if x does not freely occur in u.

The sort $\star$ denotes the universe of types and propositions, and the sort $\Box$ denotes the universe of predicate types (also called *kinds*). For instance, the type nat of natural numbers is of type $\star$, $\star$ itself is of type $\Box$ and $nat \Rightarrow \star$, the type of predicates over nat, is of type $\Box$.

We use bold face letters for denoting sequences of terms. For instance, $\boldsymbol{t}$ is the sequence $t_1 \dots t_n$ where $n = |\boldsymbol{t}|$ is the length of $\boldsymbol{t}$, and $(\boldsymbol{x} : \boldsymbol{T})U$ is the term $(x_1 : T_1) \dots (x_n : T_n)U$ (we implicitly assume that $|\boldsymbol{x}| = |\boldsymbol{T}| = n$).

We denote by $\mathrm{FV}(t)$ the set of free variables of t, by $\mathrm{dom}(\theta)$ the *domain* of a substitution θ, by $\mathrm{Pos}(t)$ the set of Dewey's positions of t, by $t|_p$ the subterm of t at position p, and by $t[u]_p$ the replacement of $t|_p$ by u.

Every symbol f is equipped with a sort s_f, an *arity* α_f and a type τ_f which may be any closed term of the form $(\boldsymbol{x} : \boldsymbol{T})U$ with $|\boldsymbol{x}| = \alpha_f$. The terms only built from variables and applications of the form $f\boldsymbol{t}$ with $|\boldsymbol{t}| = \alpha_f$ are *algebraic*.

A *typing environment* Γ is an ordered list of type declarations $\boldsymbol{x} : \boldsymbol{T}$. If f is a symbol of type $\tau_f = (\boldsymbol{x} : \boldsymbol{T})U$, we denote by Γ_f the environment $\boldsymbol{x} : \boldsymbol{T}$.

A rule for typing symbols is added to the typing rules of CC:

$$\text{(symb)} \quad \frac{\vdash \tau_f : s_f}{\vdash f : \tau_f}$$

A *rewrite rule* is a pair $l \to r$ such that (1) l is algebraic, (2) l is not a variable, and (3) $\mathrm{FV}(r) \subseteq \mathrm{FV}(l)$. Only l has to be algebraic: r may contain applications, abstractions and products. This is a particular case of Combinatory Reduction System (CRS) [18] which does not need *higher-order pattern-matching*.

If $\mathcal{G} \subseteq \mathcal{F}$, $\mathcal{R}_\mathcal{G}$ is the set of rules whose left-hand side is headed by a symbol in $\mathcal{G}$. A symbol f with $\mathcal{R}_{\{f\}} = \emptyset$ is *constant*, otherwise it is (partially) *defined*.

A rule is *left-linear* (resp. *right-linear*) if no variable occurs more than once in the left-hand side (resp. right-hand side). A rule is *linear* if it is both left-linear and right-linear. A rule is *non-duplicating* if no variable occurs more in the right-hand side than in the left-hand side.

A term t *$\mathcal{R}$-rewrites* to a term t', written $t \to_\mathcal{R} t'$, if there exists a position p in t, a rule $l \to r \in \mathcal{R}$ and a substitution σ such that $t|_p = l\sigma$ and $t' = t[r\sigma]_p$. A term t *β-rewrites* to a term t', written $t \to_\beta t'$, if there exists a position p in t such that $t|_p = ([x : U]v\ u)$ and $t' = t[v\{x \mapsto u\}]_p$. Given a relation $\to$ and a term t, let $\to(t) = \{t' \in \mathcal{T} \mid t \to t'\}$.

Finally, in CAC, $\beta\mathcal{R}$-equivalent types are identified. More precisely, in the type conversion rule of CC, $\downarrow_\beta$ is replaced by $\downarrow_{\beta\mathcal{R}}$:

$$\text{(conv)} \quad \frac{\Gamma \vdash t : T \quad T \downarrow_{\beta\mathcal{R}} T' \quad \Gamma \vdash T' : s}{\Gamma \vdash t : T'}$$

where $u \downarrow_{\beta\mathcal{R}} v$ iff there exists a term w such that $u \to^*_{\beta\mathcal{R}} w$ and $v \to^*_{\beta\mathcal{R}} w$, $\to^*_{\beta\mathcal{R}}$ being the reflexive and transitive closure of $\to_\beta \cup \to_{\mathcal{R}}$. This rule means that any term t of type T in the environment Γ is also of type T' if T and T' have a common reduct (and T' is of type some sort s). For instance, if t is a proof of $P(2+2)$ then t is also a proof of $P(4)$ if $\mathcal{R}$ contains the following rules:

$$\begin{array}{rcl} x+0 & \to & x \\ x+(s\ y) & \to & s\ (x+y) \end{array}$$

This decreases the size of proofs and increases automation as well.

A substitution θ *preserves typing from* Γ *to* Δ, written $\theta : \Gamma \rightsquigarrow \Delta$, if, for all $x \in \mathrm{dom}(\Gamma)$, $\Delta \vdash x\theta : x\Gamma\theta$, where $x\Gamma$ is the type associated to x in Γ. Type-preserving substitutions enjoy the following important property: if $\Gamma \vdash t : T$ and $\theta : \Gamma \rightsquigarrow \Delta$ then $\Delta \vdash t\theta : T\theta$.

For ensuring the *subject reduction* property (preservation of typing under reduction), every rule $f\boldsymbol{l} \to r$ is equipped with an environment Γ and a substitution ρ such that,[1] if $f : (\boldsymbol{x} : \boldsymbol{T})U$ and $\gamma = \{\boldsymbol{x} \mapsto \boldsymbol{l}\}$ then $\Gamma \vdash f\boldsymbol{l}\rho : U\gamma\rho$ and $\Gamma \vdash r : U\gamma\rho$. The substitution ρ allows to eliminate non-linearities only due to typing and thus makes rewriting more efficient and confluence easier to prove. For instance, the concatenation on polymorphic lists (type $list : \star \Rightarrow \star$ with constructors $nil : (A : \star)listA$ and $cons : (A : \star)A \Rightarrow listA \Rightarrow listA$) of type $(A : \star)listA \Rightarrow listA \Rightarrow listA$ can be defined by:

$$\begin{array}{rcl} app\ A\ (nil\ A')\ l' & \to & l' \\ app\ A\ (cons\ A'\ x\ l)\ l' & \to & cons\ A\ x\ (app\ A\ x\ l\ l') \\ app\ A\ (app\ A'\ l\ l')\ l'' & \to & app\ A\ l\ (app\ A\ l'\ l'') \end{array}$$

with $\Gamma = A : \star, x : A, l : listA, l' : listA$ and $\rho = \{A' \mapsto A\}$. For instance, $app\ A\ (nil\ A')$ is not typable in Γ (since $A' \notin \mathrm{dom}(\Gamma)$) but becomes typable if we apply ρ. This does not matter since, if an instance $app\ A\sigma\ (nil\ A'\sigma)$ is typable then $A\sigma$ is convertible to $A'\sigma$.

3 Rewriting Modulo

Now, we assume given a set $\mathcal{E}$ of *equations* $l = r$ which will be seen as a set of *symmetric* rules, that is, a set such that $l \to r \in \mathcal{E}$ iff $r \to l \in \mathcal{E}$. The conditions on rules imply that, if $l = r \in \mathcal{E}$, then (1) both l and r are algebraic, (2) both l and r are headed by a function symbol, (3) l and r have the same (free) variables.

Examples of equations are:

$$\begin{array}{rl} x+y = y+x & \text{(commutativity of } +) \\ x+(y+z) = (x+y)+z & \text{(associativity of } +) \\ x\times(y+z) = (x\times y)+(x\times z) & \text{(distributivity of } \times) \\ x+0 = x & \text{(neutrality of 0)} \end{array}$$

[1] Other conditions are necessary that we do not detail here.

$$add\ A\ x\ (add\ A'\ y\ S) = add\ A\ y\ (add\ A'\ x\ S)$$
$$union\ A\ S\ S' = union\ A\ S'\ S$$
$$union\ A\ S\ (union\ A'\ S'\ S'') = union\ A\ (union\ A'\ S\ S')\ S''$$

where $set : \star \Rightarrow \star$, $empty : (A : \star)setA$, $add : (A : \star)A \Rightarrow setA \Rightarrow setA$ and $union : (A : \star)setA \Rightarrow setA \Rightarrow setA$ formalize finite sets of elements of type A. Except for distributivity which is not linear, and the equation $x + 0 = x$ whose equivalence classes are infinite, all the other equations will satisfy our strong normalization conditions. Note however that distributivity and neutrality can always be used as rules when oriented from left to right. Hence, the word problem for abelian groups or abelian rings for instance can be decided by using *normalized rewriting* [19].

On the other hand, the following expressions are not equations since left and right-hand sides have distinct sets of variables:

$$x \times 0 = 0 \quad (0 \text{ is absorbing for } \times)$$
$$x + (-x) = 0 \quad (\text{inverse})$$

Let $\sim$ be the reflexive and transitive closure of $\rightarrow_{\mathcal{E}}$ ($\sim$ is an equivalence relation since $\mathcal{E}$ is symmetric). We are now interested in the termination of $\blacktriangleright = \rightarrow_\beta \cup \sim\rightarrow_{\mathcal{R}}$ (instead of $\rightarrow_\beta \cup \rightarrow_{\mathcal{R}}$ before). In the following, we may denote $\rightarrow_{\mathcal{E}}$ by $\mathcal{E}$, $\rightarrow_{\mathcal{R}}$ by $\mathcal{R}$ and $\rightarrow_\beta$ by β.

In order to preserve all the basic properties of the calculus, we do not change the shape of the relation used in the type conversion rule (conv): two types T and T' are convertible if $T \downarrow T'$ with $\rightarrow = \rightarrow_\beta \cup \rightarrow_{\mathcal{R}} \cup \rightarrow_{\mathcal{E}}$. But this raises the question of how to check this condition, knowing that $\rightarrow$ may be not terminating. We study this problem in Section 6.

4 Conditions of strong normalization

In the strong normalization conditions, we distinguish between *first-order* symbols (set $\mathcal{F}_1$) and *higher-order* symbols (set $\mathcal{F}_\omega$). To precisely define what is a first-order symbol, we need a little definition before. We say that a constant predicate symbol is *primitive* if it is not polymorphic and if its constructors have no functional arguments. This includes in particular any first-order data type (natural numbers, lists of natural numbers, etc.). Now, a symbol f is *first-order* if it is a predicate symbol of *maximal arity*,[2] or if it is a function symbol whose output type is a primitive predicate symbol. Any other symbol is *higher-order*. Let $\mathcal{R}_\iota = \mathcal{R}_{\mathcal{F}_\iota}$ and $\mathcal{E}_\iota = \mathcal{E}_{\mathcal{F}_\iota}$ for $\iota \in \{1, \omega\}$.

Since the pioneer works on the combination of λ-calculus and first-order rewriting [7, 20], it is well known that the addition at the object level of a strongly normalizing first-order rewrite system preserves strong normalization. This comes from the fact that first-order rewriting cannot create β-redexes. On

[2] A predicate symbol f of type $(\boldsymbol{x} : \boldsymbol{T})U$ is of *maximal arity* if $U = \star$, that is, if the elements of type $f\boldsymbol{t}$ are not functions.

the other hand, higher-order rewriting can create β-redexes. This is why we have other conditions on higher-order symbols than merely strong normalization. Furthermore, in order for the two systems to be combined without losing strong normalization [23], we also require first-order rules to be non-duplicating [21]. Note however that a first-order symbol can always be considered as higher-order (but the strong normalization conditions on higher-order symbols may not be powerful enough for proving the termination of its defining rules).

The strong normalization conditions on higher-order rewrite rules are based on the notion of *computability closure* [5]. We are going to use this notion for the equations too.

Typed λ-calculi are generally proved strongly normalizing by using Tait and Girard's technique of *computability predicates/reducibility candidates* [14]. Indeed, a direct proof of strong normalization by induction on the structure of terms does not work. The idea of Tait, later extended by Girard to the polymorphic λ-calculus, is to strengthen the induction hypothesis as follows. To every type T, one associates a set $[\![T]\!] \subseteq \mathcal{SN}$ (set of strongly normalizing terms), and proves that every term of type T is *computable*, that is, belongs to $[\![T]\!]$.

Now, if we extend such a calculus with rewriting, for preserving strong normalization, a rewrite rule has to preserve computability. The *computability closure* of a term t is a set of terms that are computable whenever t itself is computable. So, if the right-hand side r of a rule $f\boldsymbol{l} \to r$ belongs to the computability closure of $\boldsymbol{l}$, a condition called the *General Schema*, then r is computable whenever the terms in $\boldsymbol{l}$ are computable.

Formally, the computability closure for a rule $(f\boldsymbol{l} \to r, \Gamma, \rho)$ with $\tau_f = (\boldsymbol{x} : \boldsymbol{T})U$ and $\gamma = \{\boldsymbol{x} \mapsto \boldsymbol{l}\}$ is the set of terms t such that the judgment $\vdash_{\mathrm{c}} t : U\gamma\rho$ can be deduced from the rules of Figure 1, where the variables of $\mathrm{dom}(\Gamma)$ are considered as symbols ($\tau_x = x\Gamma$), $>_{\mathcal{F}}$ is a well-founded quasi-ordering (precedence) on symbols, with $x <_{\mathcal{F}} f$ for all $x \in \mathrm{dom}(\Gamma)$, $>_f$ is the multiset or lexicographic extension[3] of the subterm ordering[4] $\rhd$, and $T \downarrow_f T'$ iff T and T' have a common reduct by $\to_f = \to_\beta \cup \to_{\mathcal{R}_f^<}$ where $\mathcal{R}_f^< = \{g\boldsymbol{u} \to v \in \mathcal{R} \mid g <_{\mathcal{F}} f\}$.

In addition, every variable $x \in \mathrm{dom}(\Gamma)$ is required to be *accessible* in some l_i, that is, $x\sigma$ is computable whenever $l_i\sigma$ is computable. The arguments of a constructor-headed term are always accessible. For a function-headed term $f\boldsymbol{t}$ with $f : (\boldsymbol{x} : \boldsymbol{T})C\boldsymbol{v}$ and C constant, only the t_i's such that C occurs positively in T_i are accessible (X occurs positively in $Y \Rightarrow X$ and negatively in $X \Rightarrow Y$).

The relation $\vdash_{\mathrm{c}}$ is similar to the typing relation $\vdash$ of CAC except that symbol applications are restricted to symbols smaller than f, or to arguments smaller than $\boldsymbol{l}$ in the case of an application of a symbol equivalent to f. So, verifying that a rule satisfies the General Schema amounts to check whether r has type $U\gamma\rho$ with the previous restrictions on symbol applications. It therefore has the same complexity.

[3] Or a simple combination thereof, depending on the *status* of f.

[4] We use a more powerful ordering for dealing with recursive definitions on types whose constructors have functional arguments.

Fig. 1. Computability closure for $(f\boldsymbol{l} \to r, \Gamma, \rho)$

$$\text{(ax)} \quad \frac{}{\vdash_{\mathrm{c}} \star : \Box}$$

$$\text{(symb}^{<}\text{)} \quad \frac{\vdash_{\mathrm{c}} \tau_g : s_g}{\vdash_{\mathrm{c}} g : \tau_g} \qquad (g <_{\mathcal{F}} f)$$

$$\text{(symb}^{=}\text{)} \quad \frac{\vdash_{\mathrm{c}} \tau_g : s_g \quad \delta : \Gamma_g \rightsquigarrow_c \Delta}{\Delta \vdash_{\mathrm{c}} g\boldsymbol{y}\delta : V\delta} \qquad \begin{array}{c}(\tau_g = (\boldsymbol{y} : \boldsymbol{U})V,\\ g =_{\mathcal{F}} f \text{ and } \boldsymbol{y}\delta <_f \boldsymbol{l})\end{array}$$

$$\text{(var)} \quad \frac{\Delta \vdash_{\mathrm{c}} T : s}{\Delta, x : T \vdash_{\mathrm{c}} x : T} \qquad (x \notin \mathrm{dom}(\Delta))$$

$$\text{(weak)} \quad \frac{\Delta \vdash_{\mathrm{c}} T : s \quad \Delta \vdash_{\mathrm{c}} u : U}{\Delta, x : T \vdash_{\mathrm{c}} u : U} \qquad (x \notin \mathrm{dom}(\Delta))$$

$$\text{(abs)} \quad \frac{\Delta, x : U \vdash_{\mathrm{c}} v : V \quad \Delta \vdash_{\mathrm{c}} (x : U)V : s}{\Delta \vdash_{\mathrm{c}} [x : U]v : (x : U)V}$$

$$\text{(app)} \quad \frac{\Delta \vdash_{\mathrm{c}} t : (x : U)V \quad \Delta \vdash_{\mathrm{c}} u : U}{\Delta \vdash_{\mathrm{c}} tu : V\{x \mapsto u\}}$$

$$\text{(prod)} \quad \frac{\Delta, x : U \vdash_{\mathrm{c}} V : s}{\Delta \vdash_{\mathrm{c}} (x : U)V : s}$$

$$\text{(conv)} \quad \frac{\Delta \vdash_{\mathrm{c}} t : T \quad \Delta \vdash_{\mathrm{c}} T : s \quad \Delta \vdash_{\mathrm{c}} T' : s}{\Delta \vdash_{\mathrm{c}} t : T'} \qquad (T \downarrow_f T')$$

Now, how the computability closure can help us in dealing with rewriting modulo equations? When one tries to prove that every term is computable, in the case of a term $f\boldsymbol{t}$, it is sufficient to prove that every reduct of $f\boldsymbol{t}$ is computable. In the case of a head-reduct $f\boldsymbol{l}\sigma \to r\sigma$, this follows from the fact that r belongs to the computability closure of $\boldsymbol{l}$ since, by induction hypothesis, the terms in $\boldsymbol{l}\sigma$ are computable.

Now, with rewriting modulo, a $\mathcal{R}$-step can be preceded by $\mathcal{E}$-steps: $f\boldsymbol{t} \to_{\mathcal{E}}^{*} g\boldsymbol{u} \to_{\mathcal{R}} t'$. To apply the previous method with $g\boldsymbol{u}$, we must prove that the terms in $\boldsymbol{u}$ are computable. This can be achieved by assuming that the equations also satisfy the General Schema in the following sense: an equation $(f\boldsymbol{l} \to g\boldsymbol{m}, \Gamma, \rho)$ with $\tau_g = (\boldsymbol{x} : \boldsymbol{T})U$ and $\gamma = \{\boldsymbol{x} \mapsto \boldsymbol{m}\}$ satisfies the General Schema if, for all i, $\vdash_{\mathrm{c}} m_i : T_i\gamma\rho$, that is, the terms in $\boldsymbol{m}$ belong to the computability closure of $\boldsymbol{l}$. By symmetry, the terms in $\boldsymbol{l}$ belong to the computability closure of $\boldsymbol{m}$.

One can easily check that this condition is satisfied by commutativity (whatever the type of $+$ is) and associativity (if both y and z are accessible in $y + z$):

$$\begin{array}{rcl} x + y & = & y + x \\ x + (y + z) & = & (x + y) + z \end{array}$$

For commutativity, this is immediate and does not depend on the type of $+$: both y and x belong to the computability closure of x and y.

For associativity, we must prove that both $x + y$ and z belong to the computability closure $\mathcal{CC}$ of x and $y+z$. If we assume that both y and z are accessible in $y + z$ (which is the case for instance if $+ : nat \Rightarrow nat \Rightarrow nat$), then z belongs to $\mathcal{CC}$ and, by using a multiset status for comparing the arguments of $+$, $x + y$ belongs to $\mathcal{CC}$ too since $\{x, y\} \lhd_{\text{mul}} \{x, y + z\}$.

We now give all the strong normalization conditions.

Theorem 1 (Strong normalization of $\beta \cup \sim\mathcal{R}$). *Let $\sim_1$ be the reflexive and transitive closure of $\mathcal{E}_1$. The relation $\blacktriangleright = \rightarrow_\beta \cup \sim\rightarrow_\mathcal{R}$ is strongly normalizing if the following conditions adapted from [2] are satisfied:*

- *$\rightarrow = \rightarrow_\beta \cup \rightarrow_\mathcal{R} \cup \rightarrow_\mathcal{E}$ is confluent,*[5]
- *the rules of $\mathcal{R}_1$ are non-duplicating,*[6] *$\mathcal{R}_1 \cap \mathcal{F}_\omega = \mathcal{E}_1 \cap \mathcal{F}_\omega = \emptyset$*[7] *and $\sim_1 \rightarrow_{\mathcal{R}_1}$ is strongly normalizing on first-order algebraic terms,*
- *the rules of $\mathcal{R}_\omega$ satisfy the General Schema and are* safe,[8]
- *rules on predicate symbols have no critical pair, satisfy the General Schema*[9] *and are* small,[10]

and if the following new conditions are satisfied too:

- *there is no equation on predicate symbols,*
- *$\mathcal{E}$ is linear,*
- *the equivalence classes modulo $\sim$ are finite,*
- *every rule $(f\boldsymbol{l} \rightarrow g\boldsymbol{m}, \Gamma, \rho) \in \mathcal{E}$ satisfies the General Schema in the following sense: if $\tau_g = (\boldsymbol{x} : \boldsymbol{T})U$ and $\gamma = \{\boldsymbol{x} \mapsto \boldsymbol{m}\}$ then, for all i, $\vdash_\mathrm{c} m_i : T_i\gamma\rho$.*

Not allowing equations on predicate symbols is an important limitation. However, one cannot have equations on connectors if one wants to preserve the Curry-Howard isomorphism. For instance, with commutativity on $\wedge$, one looses subject reduction. Take $\wedge : \star \Rightarrow \star \Rightarrow \star$, $pair : (A : \star)(B : \star)A \Rightarrow B \Rightarrow A \wedge B$ and $\pi_1 : (A : \star)(B : \star)A \wedge B \Rightarrow A$ defined by $\pi_1\ A\ B\ (pair\ A'\ B'\ a\ b) \rightarrow a$. Then, $\pi_1\ B\ A\ (pair\ A\ B\ a\ b)$ is of type B but a is not.

5 Strong normalization proof

The strong normalization proof follows the one given in [6] very closely.[11] We only give the definitions and lemmas that must be modified. As previously explained,

[5] If there are type-level rewrite rules.
[6] If there are higher-order rules.
[7] First-order rules/equations only contain first-order symbols.
[8] No pattern-matching on predicates.
[9] There are other possibilities. See [2] for more details.
[10] A rule $f\boldsymbol{l} \rightarrow r$ is *small* if every predicate variable in r is equal to one of the l_i's.
[11] The proof given in [6] is an important simplification of the one given in [2].

the strong normalization is obtained by defining an interpretation $[\![T]\!] \subseteq \mathcal{SN}$ for every type T, and by proving that every term of type T belongs to $[\![T]\!]$.

More precisely, for every type T, we define the set $\mathcal{R}_T$ of the possible interpretations, or *candidates*, for the terms of type T. $\mathcal{R}_{(x:U)V}$ is the set of functions R from $\mathcal{T} \times \mathcal{R}_U$ to $\mathcal{R}_V$ that are stable by reduction: if $u \to u'$ then $R(u,S) = R(u',S)$. A term t is *neutral* if it is distinct from an abstraction or a constructor. $\mathcal{R}_\star$ is the set of sets $R \subseteq \mathcal{T}$ such that:

(R1) Strong normalization: $R \subseteq \mathcal{SN}$.

(R2) Stability by reduction: if $t \in R$ then $\to(t) \subseteq R$.

(R3) Neutral terms: if t is neutral and $\blacktriangleright(t) \subseteq R$ then $t \in R$.

Candidates form a complete lattice. A *candidate assignment* ξ is a function which associates a candidate to every variable. Given an interpretation I for predicate symbols, a candidate assignment ξ and a substitution θ, the *interpretation* of a type T, written $[\![T]\!]^I_{\xi,\theta}$, is defined in [4]. The elements of $[\![T]\!]^I_{\xi,\theta}$ are said *computable*. A pair (ξ,θ) is *Γ-valid*, written $\xi,\theta \models \Gamma$, if, for all $x \in \mathrm{dom}(\Gamma)$, $x\xi \in \mathcal{R}_{x\Gamma}$ and $x\theta \in [\![x\Gamma]\!]^I_{\xi,\theta}$.

Then, strong normalization is obtained by defining an interpretation $I_f \in \mathcal{R}_{\tau_f}$ for every predicate symbol f, and by proving that every symbol f is computable, that is, $f \in [\![\tau_f]\!]$. If $\tau_f = (\boldsymbol{x} : \boldsymbol{T})U$, it amounts to check that, for all Γ_f-valid pair (ξ,θ), $f\boldsymbol{x}\theta \in [\![U]\!]_{\xi,\theta}$. For the interpretation, we keep the one for constant predicate symbols given in [6] but slightly modify the interpretation of defined predicate symbols for taking into account the new reduction relation.

Although we do not change the interpretation of constant predicate symbols, we must check that the interpretation of *primitive* predicate symbols is still $\mathcal{SN}$ (hence that, for primitive predicate symbols, computability is equivalent to strong normalization), since this property is used for proving that a terminating and non-duplicating (if there are higher-order rewrite rules) first-order rewrite system preserves strong normalization. The verification of the former property is easy. We now prove the latter.

Lemma 2. [16] *If the $\sim$-classes are finite then $\sim\triangleright$ is strongly normalizing.*

Proof. We prove that $(\sim\triangleright)^n \subseteq \sim\triangleright^n$ by induction on n. For $n = 0$, this is immediate. For $n+1$, $(\sim\triangleright)^{n+1} \subseteq \sim\triangleright\sim\triangleright^n \subseteq \sim\sim\triangleright\triangleright^n \subseteq \sim\triangleright^{n+1}$. □

Lemma 3. [12] *If $t \in \mathcal{SN}(\beta)$ and $t \to_{\mathcal{R}_1} u$ then $\beta(t) \to^*_{\mathcal{R}_1} \beta(u)$.*

Proof. Dougherty proves this result in [12] (Proposition 4.6 and Theorem 4.7) for the untyped λ-calculus. The proof can clearly be extended to the Calculus of Algebraic Constructions. We inductively define $\twoheadrightarrow$ as follows:

- $a \twoheadrightarrow a$;
- if $l \to r \in \mathcal{R}_1$ and $\sigma \twoheadrightarrow \theta$ then $l\sigma \twoheadrightarrow r\theta$;
- if $a \twoheadrightarrow b$ and $c \twoheadrightarrow d$ then $ac \twoheadrightarrow bd$, $[x:a]c \twoheadrightarrow [x:b]d$ and $(x:a)c \twoheadrightarrow (x:b)d$;
- if $\boldsymbol{a} \twoheadrightarrow \boldsymbol{b}$ then $f\boldsymbol{a} \twoheadrightarrow f\boldsymbol{b}$.

We now prove that, if $t \to_\beta t'$ and $t \twoheadrightarrow u$ then there exist t'' and u' such that $t' \to^*_\beta t'' \twoheadrightarrow u'$ and $u \to^*_\beta u'$ by induction on $t \twoheadrightarrow u$.

- $u = t$. Immediate.
- $t = l\sigma$, $u = r\theta$ and $\sigma \twoheadrightarrow \theta$. Since left-hand sides of rules are algebraic, the β-reduction must take place in an occurrence of a variable $x \in \mathrm{FV}(l)$. Let v' be the β-reduct of $x\sigma$. By induction hypothesis, there exists v'' and w such that $v' \to_\beta^* v'' \twoheadrightarrow w$ and $x\theta \to_\beta^* w$. Let σ'' such that $x\sigma'' = v''$ and $y\sigma'' = y\sigma$ if $y \neq x$, and θ' such that $x\theta' = w$ and $y\theta' = y\theta$ if $y \neq x$. We have $\sigma'' \twoheadrightarrow \theta'$. By β-reducing all the instances of the occurrences of x in l to v'', we get $t' \to_\beta^* l\sigma'' \twoheadrightarrow r\theta'$ and, by reducing all the instances of the occurrences of x in r to w, we get $u = r\theta \to_\beta^* r\theta'$.
- Assume that $t = [x : a]c\ k$, $u = v\ l$, $[x : a]c \twoheadrightarrow v$, $k \twoheadrightarrow l$ and $t' = c\{x \mapsto k\}$. Then, $v = [x : b]d$ with $a \twoheadrightarrow b$ and $c \twoheadrightarrow d$. Therefore, $c\{x \mapsto k\} \twoheadrightarrow d\{x \mapsto l\}$ and $u \to_\beta d\{x \mapsto l\}$.
  Assume now that $t = ac$, $u = bd$, $a \twoheadrightarrow b$, $c \twoheadrightarrow d$ and $a \to_\beta a'$. The other cases are similar. By induction hypothesis, there exist a'' and b' such that $a' \to_\beta^* a'' \twoheadrightarrow b'$ and $b \to_\beta^* b'$. Therefore, $a'c \to_\beta^* a''c \twoheadrightarrow b'd$ and $bd \to_\beta^* b'd$.
- $t = f\boldsymbol{a}$, $u = f\boldsymbol{b}$ and $\boldsymbol{a} \twoheadrightarrow \boldsymbol{b}$. Then, there is i such that $t' = f\boldsymbol{a}'$, $a_i \to_\beta a_i'$ and $a_j = a_j'$ if $j \neq i$. By induction hypothesis, there exists a_i'' and b_i' such that $a_i' \to_\beta^* a_i'' \twoheadrightarrow b_i'$ and $b_i \to_\beta^* b_i'$. Let $a_j'' = a_j$ and $b_j' = b_j$ if $j \neq i$. Then, $\boldsymbol{a}'' \twoheadrightarrow \boldsymbol{b}'$, $t' = f\boldsymbol{a}' \to_\beta^* f\boldsymbol{a}'' \twoheadrightarrow f\boldsymbol{b}'$ and $u = f\boldsymbol{b} \to_\beta^* f\boldsymbol{b}'$.

Now, since t is β-strongly normalizable, we can prove the lemma by induction on $\to_\beta$. If t is in β-normal form then u also is in β-normal form since $\mathcal{R}_1$-reductions preserve β-normal forms. Hence, $\beta(t) = t \twoheadrightarrow u = \beta(u)$. Now, if $t \to_\beta t'$ then there exist t'' and u' such that $t' \to_\beta^* t'' \twoheadrightarrow u'$ and $u \to_\beta^* u'$. By induction hypothesis, $\beta(t'') \twoheadrightarrow \beta(u')$. Therefore, $\beta(t) \twoheadrightarrow \beta(u)$. □

Definition 4 (Cap and aliens). *Let ζ be an injection from the classes of terms modulo $\downarrow^*$ to $\mathcal{X}$. The* cap *of a term t is the biggest first-order algebraic term $cap(t) = t[x_1]_{p_1} \dots [x_n]_{p_n}$ such that $x_i = \zeta(t|_{p_i})$. The $t|_{p_i}$'s are called the* aliens *of t. We denote by $\beta(t)$ the β-normal form of t, by $cap\beta(t)$ the cap of $\beta(t)$, by $Cap(t)$ (resp. $Cap\beta(t)$) the $\sim_1$-equivalence class of $cap(t)$ (resp. $cap\beta(t)$), by $aliens(t)$ the multiset of the aliens of t, and by $Aliens(t)$ the multiset union of the (finite) $\sim$-equivalence classes of the aliens of t.*

Theorem 5 (Computability of first-order symbols). *If $f \in \mathcal{F}_1$ and $\boldsymbol{t} \in \mathcal{SN}$ then $f\boldsymbol{t} \in \mathcal{SN}$.*

Proof. We prove that every $\blacktriangleright$-reduct t' of $t = f\boldsymbol{t}$ is strongly normalizable. In the following, $(>_a, >_b)_{\mathrm{lex}}$ denotes the lexicographic ordering built with $>_a$ and $>_b$, and $>_{\mathrm{mul}}$ denotes the multiset extension of $>$.

Case $\mathcal{R}_\omega \neq \emptyset$. By induction on $(Aliens(t), Cap(t))$ with $((\to_\beta\sim \cup \to_{\mathcal{R}}\sim \cup \rhd\sim)_{\mathrm{mul}}, (\to_{\mathcal{R}_1}\sim_1)_{\mathrm{mul}})_{\mathrm{lex}}$ as well-founded ordering. It is easy to see that the aliens are strongly normalizable for $\to_\beta\sim$, $\to_{\mathcal{R}}\sim$ and $\rhd\sim$ since they are so for $\sim\to_\beta$ (Lemma 7), $\sim\to_{\mathcal{R}}$ and $\sim\rhd$ (Lemma 2) respectively.

If $t \to_\beta t'$ then the reduction takes place in an alien v. Let v' be its β-reduct. If v' is not headed by a symbol of $\mathcal{F}_1$ then $Aliens(t)$ $(\to_\beta\sim)_{\mathrm{mul}}$ $Aliens(u)$. Otherwise, its cap increases the cap of t' but, since the aliens of t' are then strict subterms of v', we have $Aliens(t)$ $(\to_\beta\sim \cup \rhd\sim)_{\mathrm{mul}}$ $Aliens(u)$.

Assume now that $t \to_{\mathcal{E}}^* u \to_{\mathcal{R}} t'$. We first look at what happens when $t \to_{\mathcal{E}} u$. There are two cases:

- If the reduction takes place in the cap then this is a $\mathcal{E}_1$-reduction. Since both the left-hand side and the right-hand side of a first-order rule are first-order algebraic terms, we have $cap(t) \to_{\mathcal{E}_1} cap(u)$ and, since the rules of $\mathcal{E}$ are linear, we have $aliens(t) = aliens(u)$.
- If the reduction takes place in an alien then $cap(t) = cap(u)$ and $aliens(t)$ $(\to_{\mathcal{E}})_{\mathrm{mul}}$ $aliens(u)$.

So, in both cases, $Cap(t) = Cap(u)$ and $Aliens(t) = Aliens(u)$. Therefore, by induction on the number of $\mathcal{E}$-steps, if $t \to_{\mathcal{E}}^* u$ then $Cap(t) = Cap(u)$ and $Aliens(t) = Aliens(u)$. We now look at the $\mathcal{R}$-reduction. There are two cases:

- If the reduction takes place in the cap then it is a $\mathcal{R}_1$-reduction. Since both the left-hand side and the right-hand side of a first-order rule are first-order algebraic terms, we have $cap(u) \to_{\mathcal{R}_1} cap(t')$ and, since the rules of $\mathcal{R}_1$ are non-duplicating, we have $aliens(u) \subseteq aliens(t')$. If $aliens(u) \subsetneq aliens(t')$ then $Aliens(u) \subsetneq Aliens(t')$. Otherwise, $Cap(u)$ $(\to_{\mathcal{R}_1}\sim_1)_{\mathrm{mul}}$ $Cap(t')$.
- If the reduction takes place in an alien then, as in the case of a β-reduction, we have $Aliens(t)$ $(\to_{\mathcal{R}}\sim \cup \rhd\sim)_{\mathrm{mul}}$ $Aliens(u)$.

Case $\mathcal{R}_\omega = \emptyset$. Since the t_i's are strongly normalizable and no β-reduction can take place at the top of t, t has a β-normal form. We prove that every $\blacktriangleright$-reduct t' of t is strongly normalizable, by induction on $(Cap\beta(t), Aliens(t))$ with $((\to_{\mathcal{R}_1}\sim_1)_{\mathrm{mul}}, (\to_\beta\sim \cup \to_{\mathcal{R}}\sim \cup \rhd\sim)_{\mathrm{mul}})_{\mathrm{lex}}$ as well-founded ordering.

If $t \to_\beta t'$ then $cap\beta(t) = cap\beta(t')$ and, as seen in the previous case, $Aliens(t)$ $(\to_\beta\sim \cup \rhd\sim)$ $Aliens(u)$.

Otherwise, $t \to_{\mathcal{E}}^* u \to_{\mathcal{R}_1} t'$. As seen in the previous case, $cap(t) \to_{\mathcal{E}_1}^* cap(u)$ and $Aliens(t) = Aliens(u)$. Since β and $\mathcal{E}$ commute and $\mathcal{E}$ preserves β-normal forms, we have $cap\beta(t) \to_{\mathcal{E}_1}^* cap\beta(u)$ and thus $Cap\beta(t) = Cap\beta(u)$. We now look at the $\mathcal{R}_1$-reduction. There are two cases:

- The reduction takes place in the cap. Since both the left-hand side and the right-hand side of a first-order rule are first-order algebraic terms, we have $cap(u) \to_{\mathcal{R}_1} cap(t')$ and, since β-reductions cannot reduce the cap, we have $cap\beta(u) \to_{\mathcal{R}_1} cap\beta(t')$ and thus $Cap\beta(t)$ $(\to_{\mathcal{R}_1}\sim_1)_{\mathrm{mul}}$ $Cap\beta(t')$.
- If the reduction takes place in an alien then $Aliens(t)$ $(\to_{\mathcal{R}}\sim)_{\mathrm{mul}}$ $Aliens(u)$ and, after Lemma 3, $\beta(u) \to_{\mathcal{R}_1}^* \beta(t')$. Therefore, $cap\beta(u) \to_{\mathcal{R}_1}^* cap\beta(t')$ and $Cap\beta(u)$ $(\to_{\mathcal{R}}\sim)_{\mathrm{mul}}$ $Cap\beta(t')$. □

We now come to the interpretation of defined predicate symbols. Let f be a defined predicate of type $(\boldsymbol{x} : \boldsymbol{T})U$. We define $I_f(\boldsymbol{t}, \boldsymbol{S})$ by induction on $\boldsymbol{t}, \boldsymbol{S}$ as follows. If there exists a rule $(f\boldsymbol{l} \to r, \Gamma, \rho)$ and a substitution σ such that $\boldsymbol{t} \blacktriangleright^* \sim \boldsymbol{l}\sigma$ and $\boldsymbol{l}\sigma$ is in $\blacktriangleright$-normal form, then $I_f(\boldsymbol{t}, \boldsymbol{S}) = [\![r]\!]^I_{\xi,\sigma}$ with $\sigma = \{\boldsymbol{x} \mapsto \boldsymbol{t}\}$ and $x\xi = S_{\kappa_x}$ where κ_x is given by smallness. Otherwise, we take the greatest element of $\mathcal{R}_U$.

We must make sure that the definition does not depend on the choice of the rule. Assume that there is another rule $(f\boldsymbol{l'} \to r', \Gamma', \rho')$ and a substitution σ' such that $\boldsymbol{t} \blacktriangleright^* \sim \boldsymbol{l'}\sigma'$ in normal form. By confluence and Lemma 10, we have

$\boldsymbol{l}\sigma \sim \boldsymbol{l}'\sigma'$. Since $\to$ is confluent and rules on predicate symbols have no critical pair, there exists σ'' such that $\sigma \to_{\mathcal{E}}^* \sigma''$, $\sigma' \to_{\mathcal{E}}^* \sigma''$ and $\boldsymbol{l}\sigma'' = \boldsymbol{l}'\sigma''$. Therefore, for the same reason, we must have $\boldsymbol{l} = \boldsymbol{l}'$ and $r = r'$.

Finally, we check that the interpretation is stable by reduction: if $t \to t'$ then, since $\to$ is confluent, t has a $\blacktriangleright$-normal form iff t' has a $\blacktriangleright$-normal form too.

We now prove the computability of higher-order symbols.

Theorem 6 (Computability of higher-order symbols). *If $f \in \mathcal{F}_\omega$, $\tau_f = (\boldsymbol{x} : \boldsymbol{T})U$ and $\xi, \theta \models \Gamma_f$ then $f\boldsymbol{x}\theta \in [\![U]\!]_{\xi,\theta}$.*

Proof. The proof follows the one given in [6] except that $\to$ is replaced by $\blacktriangleright$. We examine the different $\blacktriangleright$-reducts of $f\boldsymbol{x}\theta$. If this is a β-reduction, it must take place in one $x_i\theta$ and we can conclude by induction hypothesis. Otherwise, we have $f\boldsymbol{x}\theta \to_{\mathcal{E}}^* g\boldsymbol{u} \to_{\mathcal{R}} t'$. Since the equations satisfy the General Schema, the u_i's are computable. Now, if the $\mathcal{R}$-reduction takes place in one u_i, we can conclude by induction hypothesis. Otherwise, this is a head-$\mathcal{R}$-reduction and we can conclude by correctness of the computability closure. □

6 Confluence

We now study the confluence of $\to$ and the decidability of $\downarrow^*$. Let R be a relation. $\overline{R}$, R^+, R^* respectively denote the inverse, the transitive closure, and the reflexive and transitive closure of R. Composition is denoted by juxtaposition.

- R is *confluent* if $\overline{R}^* R^* \subseteq R^* \overline{R}^*$.
- R is *confluent modulo* $\sim$ or $\sim$-*confluent*[12] if $\overline{R}^* R^* \subseteq R^* \sim \overline{R}^*$.
- R is $\sim$-*confluent on* $\sim$-*classes* if $\overline{R}^* \sim R^* \subseteq R^* \sim \overline{R}^*$.
- R is *locally confluent* if $\overline{R}R \subseteq R^* \overline{R}^*$.
- R is *locally* $\sim$-*confluent* if $\overline{R}R \subseteq R^* \sim \overline{R}^*$.
- R is *locally* $\sim$-*confluent on* $\sim$-*classes* if $\overline{R} \sim R \subseteq R^* \sim \overline{R}^*$.
- R is *locally* $\sim$-*coherent* if $\mathcal{E}R \subseteq R^* \sim \overline{R}^*$.
- R and S *commute* if $\overline{R}S \subseteq S\overline{R}$.
- R $\sim$-*commutes on* $\sim$-*classes* if $\overline{R} \sim R \subseteq R \sim \overline{R}$.

Lemma 7. *If $\mathcal{E}$ is linear then $\sim$ commutes with β and $\blacktriangleright$.*

Proof. Assume that $t \to_{\beta,p} u$ (β-reduction at position p) and $t \to_{\mathcal{E},q} v$ ($\mathcal{E}$-reduction at position q). There are several cases depending on the relative positions of the different reductions.

- p and q have no common prefix. Then the reductions clearly commute and $\mathcal{E}\beta \subseteq \beta\mathcal{E}$ in this case (remember that $\overline{\mathcal{E}} = \mathcal{E}$).

[12] The definitions of confluence modulo and local confluence modulo are those of [16]. They differ from Huet's definition [15]. Huet's confluence modulo corresponds to our confluence modulo on equivalence classes, but Huet's local confluence modulo does not correspond to our local confluence modulo on equivalence classes.

- $p = q$: not possible since left-hand sides of rules are algebraic and distinct from a variable.
- $p < q$: $t|_p = [x : A]b\ a$ and $u = t[b\theta]_p$ with $\theta = \{x \mapsto a\}$.
  - Reduction in A: $v = t[[x : A']b\ a]_p$ with $A \to_{\mathcal{E}} A'$. Then, $v \to_\beta u$ and $\mathcal{E}\beta \subseteq \beta$.
  - Reduction in b: $v = t[[x : A]b'\ a]_p$ with $b \to_{\mathcal{E}} b'$. Then, $v \to_\beta t[b'\theta]_p\ {}_{\mathcal{E}}\!\leftarrow u$ and $\mathcal{E}\beta \subseteq \beta\mathcal{E}$.
  - Reduction in a: $v = t[[x : A]b\ a']_p$ with $a \to_{\mathcal{E}} a'$. Let $\theta' = \{x \mapsto a'\}$. Then, $v \to_\beta t[b\theta']_p\ {}^*_{\mathcal{E}}\!\leftarrow u$ and $\mathcal{E}\beta \subseteq \beta\mathcal{E}^*$.
- $p > q$: $t = t[l\sigma]_q$ and $v = t[r\sigma]_q$. Since left-hand sides of rules are algebraic, there is one occurrence of a variable $x \in \mathrm{FV}(l)$ such that $x\sigma \to_\beta w$. Let σ' be the substitution such that $x\sigma' = w$ and $y\sigma' = y\sigma$ if $y \neq x$. Let a (resp. b) be the number of occurrences of x in l (resp. r). Then, $u \to_\beta^{a-1} t[l\sigma']_q \to_{\mathcal{E}} t[r\sigma']_q\ {}^b_\beta\!\leftarrow v$. Since $\mathcal{E}$ is linear, we have $a = b = 1$ and thus $\mathcal{E}\beta \subseteq \beta\mathcal{E}$.

In conclusion, in every case, we have $\mathcal{E}\beta \subseteq \beta\mathcal{E}^*$. By induction on the number of $\mathcal{E}$-steps, we get $\mathcal{E}^*\beta \subseteq \beta\mathcal{E}^*$, that is, $\sim\beta \subseteq \beta\sim$. Therefore, $\sim\blacktriangleright \subseteq \blacktriangleright\sim$ since $\blacktriangleright = \beta \cup \sim\mathcal{R}$, $\sim\beta \subseteq \beta\sim \subseteq \blacktriangleright\sim$ and $\sim\sim\mathcal{R} \subseteq \blacktriangleright\sim$. □

Corollary 8. *If $\mathcal{E}$ is linear and $t \in \mathcal{SN}(\beta)$ then $t \in \mathcal{SN}(\sim\beta)$.*

Proof. Assume that $t \in \mathcal{SN}(\beta)$. We prove that $(\sim\beta)^n \subseteq \beta^n\sim$ by induction on n. For $n = 0$, this is immediate. For $n + 1$, $(\sim\beta)^{n+1} = (\sim\beta)^n \sim\beta \subseteq \beta^n \sim\sim \beta \subseteq \beta^{n+1} \sim$. Therefore, $t \in \mathcal{SN}(\sim\beta)$. □

Lemma 9. *If $\mathcal{E}$ is linear then $\to^* \subseteq \blacktriangleright^* \sim$ and $\downarrow = \blacktriangleright^* \sim {}^*\blacktriangleleft$.*

Proof. $\to^* \subseteq (\beta \cup \mathcal{E} \cup \sim\mathcal{R})^*$. Since $\sim\beta^* \subseteq \beta^*\sim$ and $\sim\sim\mathcal{R} \subseteq \sim\mathcal{R}$, we get $\to^* \subseteq \sim \cup (\sim\mathcal{R})^*\to^* \cup \beta^*\to^*$. Therefore, $\to^* \subseteq \blacktriangleright^* \sim$. □

Lemma 10. *If $\mathcal{E}$ is linear then the following propositions are equivalent: $\to$ is confluent, $\blacktriangleright$ is $\sim$-confluent, $\blacktriangleright$ is $\sim$-confluent on $\sim$-classes.*

Proof. Since $\mathcal{E}$ is linear, we have $\to^* \subseteq \blacktriangleright^* \sim$ and $\sim\blacktriangleright^* \subseteq \blacktriangleright^* \sim$. We prove that $\blacktriangleright$ is $\sim$-confluent if $\to$ is confluent: ${}^*\blacktriangleleft\blacktriangleright^* \subseteq {}^*\!\leftarrow\!\to^* \subseteq \to^* {}^*\!\leftarrow \subseteq \blacktriangleright^* \sim\sim {}^*\blacktriangleleft$. We prove that $\to$ is confluent if $\blacktriangleright$ is $\sim$-confluent: ${}^*\!\leftarrow\!\to^* \subseteq \sim\ {}^*\blacktriangleleft\blacktriangleright^*\sim \subseteq \sim\blacktriangleright^* \sim {}^*\blacktriangleleft \sim \subseteq \blacktriangleright^* \sim\sim\sim {}^*\blacktriangleleft$. We now prove that $\blacktriangleright$ is $\sim$-confluent on $\sim$-classes if $\blacktriangleright$ is $\sim$-confluent (the inverse is trivial): ${}^*\blacktriangleleft \sim \blacktriangleright^* \subseteq {}^*\blacktriangleleft\blacktriangleright^* \sim \subseteq \blacktriangleright^* \sim {}^*\blacktriangleleft \sim \subseteq \blacktriangleright^* \sim\sim {}^*\blacktriangleleft$. □

Theorem 11. *Type-checking is decidable if $\blacktriangleright$ is weakly normalizing, $\mathcal{R}$ is finitely branching, $\blacktriangleright$ is $\sim$-confluent on $\sim$-classes, $\mathcal{E}$ is linear and $\sim$ is decidable.*

Proof. Type-checking is deciding whether a term t has type T in an environment Γ. A type for t can be easily inferred. Then, one checks that it is equivalent to T (see [10] for more details). Thus, we are left to prove that $\downarrow^*$ is decidable. Since $\mathcal{E}$ is linear and $\blacktriangleright$ is $\sim$-confluent on $\sim$-classes, by Lemma 10, $\to$ is confluent and $\downarrow^* = \downarrow$. Since $\mathcal{E}$ is linear, by Lemma 9, $\downarrow = \blacktriangleright^* \sim {}^*\blacktriangleleft$. Since $\blacktriangleright$ is weakly normalizing

and finitely branching ($\sim$-classes are finite and β and $\mathcal{R}$ are finitely branching), one can define a function nf computing a $\blacktriangleright$-normal form of a term. We prove that $t \downarrow^* u$ only if $nf(t) \sim nf(u)$ (the inverse is trivial). Assume that $t \blacktriangleright^* t' \sim u' {}^*\!\blacktriangleleft u$. Since $\blacktriangleright$ is $\sim$-confluent on $\sim$-classes, $nf(t) \sim nf(t') {}^*\!\blacktriangleleft t' \sim u' \blacktriangleright^* nf(u') \sim nf(u)$. Again, since $\blacktriangleright$ is $\sim$-confluent on $\sim$-classes, there exist t'' and u'' such that $nf(t) \sim nf(t') \blacktriangleright^* t'' \sim u'' {}^*\!\blacktriangleleft nf(u') \sim nf(u)$. Since $nf(t')$ and $nf(u')$ are $\blacktriangleright$-normal forms, we have $nf(t) \sim nf(u)$. □

Lemma 12. *For all relation R, if R $\sim$-commutes on $\sim$-classes then $\sim R$ is $\sim$-confluent on $\sim$-classes.*

Proof. Let $S = \sim R$. We prove that $\overline{S}^p \sim S^n \subseteq S^n \sim \overline{S}^p$ by induction on n.

- Case $n = 0$. By induction on p. The case $p = 0$ is immediate. Case $p+1$: $\overline{S}^{p+1} \sim = \overline{S}\overline{S}^p \sim \subseteq \overline{S} \sim \overline{S}^p \subseteq \sim \overline{S}\overline{S}^p$ since $\overline{S} \sim = \overline{R} \sim\sim = \overline{R} \sim = \overline{S} \subseteq \sim \overline{S}$.
- Case $n = 1$. By induction on p.
  - Case $p = 0$. $\sim S = \sim\sim R = \sim R = S \subseteq S \sim$.
  - Case $p+1$. $\overline{S}^{p+1} \sim S = \overline{S}\overline{S}^p \sim S \subseteq \overline{S}S \sim \overline{S}^p \subseteq S \sim \overline{S}\overline{S}^p$ since $\overline{S}S \sim = \overline{R} \sim\sim R \sim = \overline{R} \sim R \sim \subseteq R \sim \overline{R} \sim \subseteq S \sim \overline{S}$.
- Case $n+1$. $\overline{S}^p \sim S^{n+1} = \overline{S}^p \sim SS^n \subseteq S \sim \overline{S}^p S^n \subseteq S \sim \overline{S}^p \sim S^n \subseteq S \sim S^n \sim \overline{S}^p$ and we prove that $S \sim S^n \sim \subseteq S^{n+1} \sim$ by induction on n. The case $n = 0$ is immediate. Case $n+1$: $S \sim S^{n+1} \sim \subseteq S \sim S^n \sim S \sim \subseteq S^{n+1} \sim S \sim \subseteq S^{n+1} S \sim$ since $\sim S = \sim\sim R = \sim R = S$. □

Lemma 13. *For all relation R, if R is $\sim$-confluent on $\sim$-classes then $\sim R$ is $\sim$-confluent on $\sim$-classes.*

Proof. If R is $\sim$-confluent on $\sim$-classes then R^* $\sim$-commutes on $\sim$-classes. Hence, by Lemma 12, $\sim R^*$ is $\sim$-confluent on $\sim$-classes. Therefore, $\sim R$ is $\sim$-confluent on $\sim$-classes since $(\sim R)^* \subseteq (\sim R^*)^*$ and $(\sim R^*)^* \subseteq (\sim R)^* \sim$. □

Theorem 14. *$\blacktriangleright$ is $\sim$-confluent on $\sim$-classes if $\blacktriangleright$ is strongly normalizing, $\mathcal{E}$ is linear, $\mathcal{R}$ is locally $\sim$-confluent and $\mathcal{R}$ is locally $\sim$-coherent.*

Proof. We first prove that $\beta \cup \mathcal{R}$ is $\sim$-confluent on $\sim$-classes. In [15], Huet proves that a relation R is $\sim$-confluent on $\sim$-classes if $R \sim$ is strongly normalizing, R is locally $\sim$-confluent and R is locally $\sim$-coherent. We take $R = \beta \cup \mathcal{R}$ and check the conditions. $R\sim$ is strongly normalizing since $\blacktriangleright$ is strongly normalizing and β and $\sim$ commute ($\mathcal{E}$ is linear). Local confluence: $\overline{\beta}\beta \subseteq \beta^*\overline{\beta}^*$ since β is locally confluent, $\overline{\mathcal{R}}\beta \subseteq \beta^*\overline{\mathcal{R}}^*\overline{\beta}^*$ after the proof of Lemma 7, and $\overline{\mathcal{R}}\mathcal{R} \subseteq \mathcal{R}^* \sim \overline{\mathcal{R}}^*$ by assumption. Local coherence: $\mathcal{E}\beta \subseteq \beta\mathcal{E} \subseteq \beta \sim$ since $\mathcal{E}$ is linear, and $\mathcal{E}\mathcal{R} \subseteq \mathcal{R}^* \sim \overline{\mathcal{R}}^*$ by assumption.

So, $R = \beta \cup \mathcal{R}$ is $\sim$-confluent on $\sim$-classes. Therefore, by Lemma 13, $\sim R$ is $\sim$-confluent on $\sim$-classes. We now prove the theorem. We have $\blacktriangleright^* \subseteq (\sim R)^*$ and $(\sim R)^* \subseteq \blacktriangleright^* \sim$ (β and $\sim$ commute since $\mathcal{E}$ is linear). Thus, ${}^*\!\blacktriangleleft \sim \blacktriangleright^* \subseteq (\overline{\sim R})^* \sim (\sim R)^* \subseteq (\sim R)^* \sim (\overline{\sim R})^* \subseteq \blacktriangleright^* \sim\sim {}^*\!\blacktriangleleft$. □

Huet also proves in [15] that $\mathcal{R}$ is locally $\sim$-confluent iff its critical pairs are $\sim$-confluent, and that $\mathcal{R}$ is locally $\sim$-coherent if $\mathcal{R}$ is left-linear and the critical pairs between $\mathcal{R}$ and $\mathcal{E}$ are $\sim$-confluent. So, $\sim$-confluence is decidable whenever $\blacktriangleright$ is strongly normalizing, $\sim$ is decidable and $\mathcal{R} \cup \mathcal{E}$ is finite: it amounts to checking whether the critical pairs between the rules, and between the rules and the equations (in both directions), are $\sim$-confluent.

Unfortunately, when considering type-level rewriting, confluence is required for proving strong normalization. Whether strong normalization can be proved by using local confluence only is an open problem. Fortunately, confluence can be proved for a large class of rewrite systems without using strong normalization, namely the left-linear systems.

Theorem 15. *$\blacktriangleright$ is $\sim$-confluent on $\sim$-classes if $\mathcal{E}$ is linear, $\mathcal{R}$ is left-linear and $\mathcal{R}$ is $\sim$-confluent on $\sim$-classes.*

Proof. In [24], Van Oostrom and Van Raamsdonk prove that the combination of two left-linear and confluent Combinatory Reduction Systems (CRS) $\mathcal{H}$ and $\mathcal{J}$ is confluent if all the critical pairs between the rules of $\mathcal{H}$ and the rules of $\mathcal{J}$ are trivial. We prove the theorem by taking $\mathcal{H} = \mathcal{R} \cup \mathcal{E}$ and $\mathcal{J} = \beta$, and by proving that $\mathcal{H}$ is confluent. Since $\mathcal{H}^* \subseteq (\sim\mathcal{R})^* \sim$, we have $\overline{\mathcal{H}}^* \mathcal{H}^* \subseteq \sim (\overline{\sim\mathcal{R}})^* (\sim\mathcal{R})^* \sim$. Since $\mathcal{R}$ is $\sim$-confluent on $\sim$-classes, by Lemma 13, $\sim\mathcal{R}$ is $\sim$-confluent on $\sim$-classes. Therefore, $\sim (\overline{\sim\mathcal{R}})^* (\sim\mathcal{R})^* \sim \subseteq \sim (\sim\mathcal{R})^* \sim (\overline{\sim\mathcal{R}})^* \sim \subseteq \mathcal{H}^* \overline{\mathcal{H}}^*$. □

Again, $\mathcal{R}$ is $\sim$-confluent on $\sim$-classes if $\sim\mathcal{R}$ is strongly normalizing and $\mathcal{R}$ is locally confluent and $\sim$-coherent, which can be proved by analyzing the critical pairs between the rules and between the rules and the equations (when $\mathcal{R}$ is left-linear) [15].

7 Conclusion

In [3, 2], we give general syntactic conditions based on the notion of computability closure for proving the strong normalization of β-reduction and (higher-order) rewriting. In this paper, we show that the notion of computability closure can also be used for proving the strong normalization of β-reduction and (higher-order) rewriting modulo (higher-order) equations. It is interesting to note that, in our approach, the introduction of equations does not affect the conditions on rules: although based on the same notion, equations and rules are dealt with separately. Finally, one may wonder whether our method could be extended to Jouannaud and Rubio's Higher-Order Recursive Path Ordering (HORPO) [17, 25], which also uses the notion of computability closure for increasing its expressive power.

Acknowledgments. I thank J.-P. Jouannaud, F. van Raamsdonk and the referees for their useful comments on previous versions of this paper. Part of this work was performed during my stay at Cambridge (UK) thanks to a grant from the INRIA.

References

1. H. Barendregt. Lambda calculi with types. In S. Abramski, D. Gabbay, and T. Maibaum, editors, *Handbook of logic in computer science*, volume 2. Oxford University Press, 1992.
2. F. Blanqui. *Théorie des Types et Récriture*. PhD thesis, Université Paris XI, Orsay, France, 2001. Available in english as "Type Theory and Rewriting".
3. F. Blanqui. Definitions by rewriting in the Calculus of Constructions (extended abstract). In *Proc. of LICS'01*.
4. F. Blanqui. Definitions by rewriting in the Calculus of Constructions, 2002. Journal submission, 68 pages.
5. F. Blanqui, J.-P. Jouannaud, and M. Okada. Inductive-data-type Systems. *Theoretical Computer Science*, 272:41–68, 2002.
6. F. Blanqui. A short and flexible strong normalization proof for the Calculus of Algebraic Constructions with curried rewriting, 2003. Draft.
7. V. Breazu-Tannen and J. Gallier. Polymorphic rewriting conserves algebraic strong normalization. In *Proc. of ICALP'89*, LNCS 372.
8. E. Contejean, C. Marché, B. Monate, and X. Urbain. CiME, 2000.
9. T. Coquand and G. Huet. The Calculus of Constructions. *Information and Computation*, 76(2–3):95–120, 1988.
10. T. Coquand. An algorithm for testing conversion in type theory. In G. Huet, G. Plotkin, editors, *Logical Frameworks*, p. 255–279. Cambridge Univ. Press, 1991.
11. N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, vol. B, chap. 6. North-Holland, 1990.
12. D. Dougherty. Adding algebraic rewriting to the untyped lambda calculus. *Information and Computation*, 101(2):251–267, 1992.
13. G. Dowek, T. Hardin, and C. Kirchner. Theorem proving modulo. Technical Report 3400, INRIA Rocquencourt, France, 1998.
14. J.-Y. Girard, Y. Lafont and P. Taylor. *Proofs and Types*. Cambridge University Press, 1988.
15. G. Huet. Confluent reductions: Abstract properties and applications to term-rewriting systems. *Journal of the ACM*, 27(4):797–821, 1980.
16. J.-P. Jouannaud and H. Kirchner. Completion of a set of rules modulo a set of equations. *SIAM Journal on Computing*, 15(4):1155–1194, 1986.
17. J.-P. Jouannaud and A. Rubio. The Higher-Order Recursive Path Ordering. In *Proc. of LICS'99*.
18. J. W. Klop, V. van Oostrom, F. van Raamsdonk. Combinatory reduction systems: introduction and survey. *Theoretical Computer Science*, 121:279–308, 1993.
19. C. Marché. Normalised rewriting and normalised completion. In *Proc. of LICS'94*.
20. M. Okada. Strong normalizability for the combined system of the typed lambda calculus and an arbitrary convergent term rewrite system. In *Proc. of ISSAC'89*.
21. M. Rusinowitch. On termination of the direct sum of term-rewriting systems. *Information Processing Letters*, 26(2):65–70, 1987.
22. Coq Development Team. *The Coq Proof Assistant Reference Manual – Version 7.4*. INRIA Rocquencourt, France, 2003. `http://coq.inria.fr/`.
23. Y. Toyama. Counterexamples to termination for the direct sum of term rewriting systems. *Information Processing Letters*, 25(3):141–143, 1987.
24. V. van Oostrom and F. van Raamsdonk. Weak orthogonality implies confluence: the higher-order case. In *Proc. of LFCS'94*, LNCS 813.
25. D. Walukiewicz-Chrząszcz. Termination of rewriting in the Calculus of Constructions. *Journal of Functional Programming*, ?(?):?–?, 2002.